Mode function from here:
https://stackoverflow.com/questions/2547402/is-there-a-built-in-function-for-finding-the-mode

Introduction

This report goes through an image recognition problem. The dataset given was the fashion MNIST dataset which contains a large number of images for different types of apparel. The original dataset represent the images with 784 (28 by 28) grayscale pixels. For the purposes of this project, the data were condensed to 49 (7 by 7) pixels. The training set consisted of 60,000 labeled images and test set consisted of 10,000 labeled images. Each row was a flattened (49 by 1) version of the image with a single pixel value per column. The possible labels for each image are: Ankle boot, Bag, Coat, Dress, Pullover, Sandal, Shirt, Sneaker, T-shirt/top, and Trouser.

The machine learning methods we decided to try were: multinomial logistic regression, k-nearest neighbors, classification tree, ridge regression, lasso regression, SVM, random forest, XGBoost, and our own ensemble method.

Multinomial Logistic Regression

Multinomial logistic regression is a linear classification model. In the multinomial case, it models the log-odds of a data point being in some class k as a line. It predicts the most probable class for a particular point as the label of that point. An advantage of using this classifier is that we can actually recover the probability of each point belonging to a certain class, which is not something we can do as easily with non-linear classifiers such as decision trees or K-nearest neighbors. A disadvantage of this model is that it assumes a linear decision boundary which can be restrictive, depending on the data. It can be made more powerful if we consider feature spaces that are in higher dimensions than the original feature space of the data, but higher feature spaces don't make a lot of sense for image data. Especially since the number of feature spaces is already quite high with images and we don't want to blow up the data too much.

There were not really any parameters to tune for multinomial logistic regression. In some packages or in other languages, you can choose to regularize the regression model. Regularization works well if your model is overfitting your data or if there are highly correlated variables in your dataset (collinearity). Here, we kept the model simple as the **nnet** package doesn't provide many parameter tuning options for logistic regression. We chose this model because it's a great baseline model since it's linear and there's not much tuning to be done. We should be worried if we do worse than this model and we should aim to do quite a bit better than this model. We see that the test error for this model ranged from about 29% to 24% depending on the training set size (error decreased with training set size, as can be expected with most models until adding more data has no effect). As reference, random guessing would give us a test error around 90%. Therefore, this is a large improvement and the model is learning patterns from the data.

RandomForest

Random forests are a type of ensemble method that work for both classification and regression. They use bootstrapped samples of decision trees and introduce randomness. They are ‘random’ because each time one of the trees makes a split, it only considers a subset of the features. For example, if our data has 10 features, a random forest will only consider a random sample of 5 of those features to split on for each node split. This is an advantage because it helps decrease correlation in trees within the forest. This is important because ensembling methods really only work if the models you are combining are making different mistakes, otherwise you’re not getting the advantages of combining methods. Furthermore, random forests rarely overfit the data which is another major advantage that they have over decision trees. A disadvantage of random forests is that they are not very interpretable, unlike multinomial logistic regression. They don’t easily give you the probabilities of a data point being in a particular class and it’s not always evident how a single feature is affecting the way random forest is making a prediction.

There are several parameters that you can tune in a random forest. For example, you can tune the maximum depth of each decision tree, the number of trees in the forest (although really, more is always better with forests), or the number of features to consider at each split. We decided to only tune the number of features considered at each split for this project. We performed a grid search on values from 1 to half the number of columns (25). The value that gave us the best cross-validation results was 8. It’s interesting to note that this is very close to the default value that R would set as the number of features to consider, which is the square root of the number of features. The random forest with a training set size of 5000 (the largest size we used) had the third lowest test error, so it performs very well.

XGBoost (Gradient Boosting)

Gradient boosting is another type of ensemble method (like random forests) but that doesn’t use bagging like random forests do. It also uses decision trees, however it generally uses decision-stumps or very short trees. These are weak classifiers which are considered weak as they barely do better than random guessing. Better classifiers can also be used in gradient boosting but boosting weak classifiers usually gets you only slightly worse results and boosting will run much faster. Each classifier is learned sequentially, meaning that the errors of the first classifier are fed into the second, the errors of the second classifier are fed into the third, and so on. XGBoost usually performs as well, if not better, than random forests. One of their disadvantages is that they take longer to train than random forests because each classifier needs to be built sequentially rather than in parallel. They also have the disadvantage of not being as interpretable as something like a linear classifier, similar to the disadvantage of random forests.

Like random forests, there are quite a few parameters to tune in gradient boosting. For example, we can again tune the maximum depth of each tree, the learning rate, and the number of trees. For this project, we decided to tune the maximum depth and the learning rate. The grid search resulted in a maximum depth of 7 and a learning rate of 0.1 as best. A depth of 7 is quite a bit larger than a decision stump but since we did not perform the grid search with time to train the model in mind, 7 seems reasonable. The XGBoost model trained on 5000 data points gave us the second lowest testing error, only beat out by our ensemble method (that uses both XGBoost and random forests).

KNN - 10.

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.Training a KNN model is simply storing all of the training data. The real work happens when we want to make a prediction. To make a prediction, KNN will find the "K" nearest neighbors of a test point based on some distance metric (eg. Euclidean distance). In regression, KNN will simply average the "y" values of the k closest points for a given test point. In classification, KNN will take the majority vote of the "y" values of the k closest points for a given test point. If k = 1, then the object is simply assigned to the class of that single nearest neighbor. Lower k values mean a higher complexity of the model. If we were to plot the decision boundaries made with low k values, we would get very irregular boundaries compared to if we chose a higher k. An advantage of the model is that they generally work well for both classification and regression and it's easy to understand what they're doing. A major disadvantage is that they don't work well in very high feature spaces because the notion of what points are 'close' together doesn't translate well to higher dimensions. Scaling the data can help so that all features are on similar scales, but the problem remains. We selected this technique because it is a good baseline for non-linear and non-parametric models.

The only parameter tuning to be done with KNN is the value k. We chose to try two different k values and count these as two different models. In the future, we would do a grid-search on several k values, say, from 1 to 50, and select the k value that gave us the best cross-validation results. KNN with 10 neighbors gave us a test error of 20%, which is similar to the regression models (not including multinomial logistic regression).

KNN - 5

Compared with the KNN – 10 above, we don't see much of a difference in the running time for KNN – 5 or the test errors. The lack of difference in running time is not too surprising assuming that R calculates the K nearest neighbors in an intelligent way. That is to say, if R calculates the distances of each test point from every training point and sorts them, this takes the same amount of time regardless of the value of K. To get the K 'nearest' ones is then simply indexing the K first values of these sorted neighbors. The lack of significant difference in test error is likely due to the fact that the data set is much larger than 5 or 10 so a difference in 5 neighbors is not very noticeable. As mentioned above, in the future we would do a grid search to find the best value for k.

Reference:
https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
Title: Machine Learning Basics with the K-Nearest Neighbors Algorithm
Author: Onel Harrison
Classification Tree

Classification trees are another type of non-parametric and non-linear model. Classification trees work by first having all the data in a single root node. Then it searches through all the features and all possible "split" points and choose to split on the feature that leads to the greatest decrease in impurity. For example, a tree may have a node that splits the data in that node based on whether feature 5 is greater than 10. This decision was chosen because the resulting two nodes have lower impurity than the nodes they were split from. Classification trees are great because they don't care about the scale of your data and they don't care about the scale of your data and work on many different kinds of data. The main disadvantage of decisions trees is that they tend to overfit the training data if you do not control their growth. Decision trees can achieve 0% training error because you can always keep splitting the data somehow until each leaf has only one data point. The problem with this is that this will not generalize well to new data since we only learned all the nuances of the test data. There are some strategies to control this overfitting such as controlling the maximum depth of the tree or using a technique called "post-pruning". Furthermore, if the training data is slightly changed (still with the same columns but different rows), a completely different classification tree can results. Classification trees are quite unstable if you only use one. We selected this technique because it's a model that is easy to understand how it works and so that we can compare it with a random forest which we try below.

As mentioned just above, we can tune the maximum depth of the tree as well as, for example, the maximum number of leaves in the tree or the minimum number of samples needed in a node to consider splitting it. For this project, we decided to use the default settings. Compared with the error results of the other models, the Classification Tree didn't perform so well. This is probably due to the fact that it's overfitting the training data.

Ref:
https://ocw.mit.edu/courses/sloan-school-of-management/15-062-data-mining-spring-2003/lecture-notes/L3ClassTrees
Title: Classification and Regression Tree
Author:Prof. Nitin Patel

Ridge and lasso reference:
https://www-bcf.usc.edu/~gareth/ISL/ISLR%20First%20Printing.pdf
Title: An Introduction to Statistical Learning
Author: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

Ridge:
The ridge regression method fits the model by minimizing the following equation:
\[\sum_{i = 1}^n (y_i -\beta_0 - \sum_{j = 1}^p \beta_j x_{ij})^2 + \lambda \sum_{j=1}^p \beta_j^2 = RSS + \lambda \sum_{j=1}^p \beta_j^2 \]
This is similar to least squares for linear regression except there is a penalty for the L2 norm of each weight. The major advantage of this method is that it can shrink the value of estimated coefficients toward 0 which prevents overfitting. Even though the learned coefficients will go towards 0, they will not actually be 0 and so all features still contribute something to the model. The $\lambda \sum_{j=1}^p \beta_j^2$ term is called a shrinkage penalty. A disadvantage of this model is that it is a linear model which can be restrictive when our data is not linearly separable, and this data most likely isn't since it's high dimensional image data. We chose this method because we wanted to see how a basic regression model would fare on a fairly complicated dataset.
To determine which value of λ will be best for our model, we use the cross-validation method. We use the cv.glmnet() function to find the best λ that leads to the smallest error on the cross-validation folds. Then, we use that λ value to predict the labels of the test set. What's interesting is that we don't see a large difference in test error with change in sample size for the three ridge regression models. Surprisingly, at least to us, this model performed fairly well, getting only about 22% test error, which is quite lower than the classification tree and even lower than the multinomial logistic regression model.

Lasso:
The Lasso method is very similar to ridge regression. The only difference is that the shrinkage penalty $\lambda \sum_{j=1}^p \beta_j^2$ is replaced by $\lambda \sum_{j=1}^p |\beta_j|$. As we mentioned above, ridge regression will keep all features we include in the model. The big advantage of lasso compared to ridge is that it can perform feature selection. In other words, the

lasso method will make coefficients of some features exactly equal to 0. We chose this method because we wanted to compare the L1 vs L2 penalization between Lasso and Ridge regression. Similar to ridge, we used the same method for lasso to determine the best value of λ. Lasso performs a bit better than Ridge, the test errors are down to 20% rather than 22%. Unlike Ridge, we do see a decrease in test error with larger sample size, similar to what we see in almost all of the other models.

SVM reference:
https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
Title: SVM (Support Vector Machine) — Theory
Author: Savan Patel

SVM:
The SVM algorithm is a very useful tool for classification. It classifies the data points by finding a hyperplane that maximizes the margin of the d-dimensional space, where d is the dimension of the training data. An advantage of SVMs is that they're not only limited to linear decision boundaries. We can specify different kernels in the svm() function to set non-linear boundaries. Kernels map the input feature space to higher-dimensional feature spaces which allows is to get much more complicated decision boundaries. The main disadvantage is that there are different hyperparameters that we have to tune in order to get the best results, along with choosing the right kernel. We chose this technique because like multinomial logistic regression, it is a good model to use as a baseline.

In this problem, we used the radial kernel function to fit the model. The SVM actually performed quite well compared to many of the other classifiers we tried. It had the fourth lowest test error on the run with the largest sample size.

Ensemble
Ensemble methods combine the predictions of several different classifiers in some way to make new predictions. As discussed above, random forests and XGBoost are examples of ensemble methods that make use of decisions trees. Ensemble methods don't necessarily have to combine the same family of classifier. For example, they could make use of a decision tree and logistic regression. The major advantage, and motivation, for using ensemble methods is that the classifiers that you average are making different mistakes and therefore averaging eliminates those mistakes. For example, if you are combining the results of a random forest and an SVM,

your hope is that the random forest will correctly predict the points that the SVM got wrong and vice versa. This will occur more often if the classifiers are uncorrelated. A disadvantage is that there are many options of ways to write an ensemble method so there are many choices to be made when building one and various factors to consider.

For this project, we created a fairly simple ensemble method. We gave our ensemble method the predictions of our three “best” classifiers (random forest, SVM, and XGBoost), and simply took the most common prediction of the three as the prediction for the ensemble. This a simple procedure yet it did give us the best results. The ensemble method always gave the lowest test error of all the models when comparing models with the same sample size.